Category Theoretic Semantics for Logic Programming: Laxness and Saturation

John Power
Department of Computer Science
University of Bath
Bath, BA2 7AY, UK*
A.J.Power@bath.ac.uk

1 Summary

Recent research on category theoretic semantics of logic programming has focused on two ideas: lax semantics [3] and saturated semantics [1]. Until now, the two ideas have been presented as alternatives, but that competition is illusory, the two ideas being two views of a single, elegant body of theory, reflecting different but complementary aspects of logic programming.

Given a set of atoms At, one can identify a variable-free logic program P built over At with a P_fP_f-coalgebra structure on At, where P_f is the finite powerset functor on Set: each atom is the head of finitely many clauses in P, and the body of each clause contains finitely many atoms. If $C(P_fP_f)$ is the cofree comonad on P_fP_f, then, given a logic program P qua P_fP_f-coalgebra, the corresponding $C(P_fP_f)$-coalgebra structure characterises the and-or derivation trees generated by P.

Extending this to arbitrary programs, given a signature Σ of function symbols, let $\mathscr{L}_\Sigma$ denote the Lawvere theory generated by Σ, and, given a logic program P with function symbols in Σ, consider the functor category $[\mathscr{L}_\Sigma^{op}, Set]$, extending the set At of atoms in a variable-free logic program to the functor from $\mathscr{L}_\Sigma^{op}$ to Set sending a natural number n to the set $At(n)$ of atomic formulae with at most n variables generated by the function symbols in Σ and the predicate symbols in P. We would like to model P by a $[\mathscr{L}_\Sigma^{op}, P_fP_f]$-coalgebra $p : At \longrightarrow P_fP_fAt$ that, at n, takes an atomic formula $A(x_1,\ldots,x_n)$ with at most n variables, considers all substitutions of clauses in P into clauses with variables among $x_1,\ldots,x_n$ whose head agrees with $A(x_1,\ldots,x_n)$, and gives the set of sets of atomic formulae in antecedents. However, that does not work for two reasons. The first may be illustrated as follows.

Example 1 *ListNat (for lists of natural numbers) denotes the logic program*
1. $\mathtt{nat(0)} \leftarrow$
2. $\mathtt{nat(s(x))} \leftarrow \mathtt{nat(x)}$
3. $\mathtt{list(nil)} \leftarrow$
4. $\mathtt{list(cons(x,y))} \leftarrow \mathtt{nat(x)}, \mathtt{list(y)}$

ListNat has nullary function symbols `0` *and* `nil`. *So there is a map in* $\mathscr{L}_\Sigma$ *of the form* $0 \to 1$ *that models the function symbol* 0. *Naturality of* $p : At \longrightarrow P_fP_fAt$ *in* $[\mathscr{L}_\Sigma^{op}, Set]$ *would yield commutativity of the*

*John Power would like to acknowledge the support of EPSRC grant EP/K028243/1 and Royal Society grant IE151369. No data was generated in the course of this project.

Submitted to:
CoALPTy 2016

diagram

$$\begin{array}{ccc} At(1) & \xrightarrow{p_1} & P_fP_fAt(1) \\ \Big\downarrow{\scriptstyle At(0)} & & \Big\downarrow{\scriptstyle P_fP_fAt(0)} \\ At(0) & \xrightarrow{p_0} & P_fP_fAt(0) \end{array}$$

But consider $\mathtt{nat(x)} \in At(1)$*: there is no clause of the form* $\mathtt{nat(x)} \leftarrow$ *in ListNat, so commutativity of the diagram would imply that there cannot be a clause in ListNat of the form* $\mathtt{nat(0)} \leftarrow$ *either, but in fact there is one. Thus p is not a map in the functor category* $[\mathscr{L}_\Sigma^{op}, Set]$.

Lax semantics addresses this by relaxing the naturality condition on p to a subset condition, so that, given, for instance, a map in $\mathscr{L}_\Sigma$ of the form $f : 0 \to 1$, the diagram need not commute, but rather the composite via $P_fP_fAt(1)$ need only yield a subset of that via $At(0)$. In contrast, saturation semantics works as follows. Regarding $ob(\mathscr{L}_\Sigma)$, equally $ob(\mathscr{L}_\Sigma)^{op}$, as a discrete category with inclusion functor $I : ob(\mathscr{L}_\Sigma) \longrightarrow \mathscr{L}_\Sigma$, the functor

$$[I, Set] : [\mathscr{L}_\Sigma^{op}, Set] \longrightarrow [ob(\mathscr{L}_\Sigma)^{op}, Set]$$

that sends $H : \mathscr{L}_\Sigma^{op} \longrightarrow Set$ to the composite $HI : ob(\mathscr{L}_\Sigma)^{op} \longrightarrow Set$ has a right adjoint R, given by right Kan extension. So the data for $p : At \longrightarrow P_fP_fAt$ may be seen as a map in $[ob(\mathscr{L}_\Sigma)^{op}, Set]$, which, by the adjointness, corresponds to a map $\bar{p} : At \longrightarrow R(P_fP_fAtI)$ in $[\mathscr{L}_\Sigma^{op}, Set]$, yielding saturation semantics. In this talk, we show that the two approaches can elegantly be unified, the relationship corresponding to the relationship between theorem proving and proof search in logic programming.

The second problem mentioned above is about *existential* variables, which we now illustrate.

Example 2 *GC (for graph connectivity) denotes the logic program*
1. $\mathtt{connected(x,x)} \leftarrow$
2. $\mathtt{connected(x,y)} \leftarrow \mathtt{edge(x,z)}, \mathtt{connected(z,y)}$

There is a variable z in the tail of the second clause of GC that does not appear in its head. Such a variable is called an existential variable, the presence of which challenges the algorithmic significance of lax semantics. In describing the putative coalgebra $p : At \longrightarrow P_fP_fAt$ just before Example 1, we referred to *all* substitutions of clauses in P into clauses with variables among $x_1, \ldots, x_n$ whose head agrees with $A(x_1, \ldots, x_n)$. If there are no existential variables, that amounts to term-matching, which is algorithmically efficient; but if existential variables do appear, the mere presence of a unary function symbol generates an infinity of such substitutions, creating algorithmic difficulty, which, when first introducing lax semantics, we, also Bonchi and Zanasi, avoided modelling by replacing the outer instance of P_f by P_c, thus allowing for countably many choices. Such infiniteness militates against algorithmic efficiency, and we resolve it by refining the functor P_fP_f while retaining finiteness.

This talk is based upon the paper [2].

References

[1] Filippo Bonchi & Fabio Zanasi (2015): *Bialgebraic Semantics for Logic Programming*. *Logical Methods in Computer Science* 11(1).

[2] Ekaterina Komendantskaya & John Power (2016): *Logic programming: laxness and saturation*. Submitted, *CoRR* abs/1608.07708. Available at `http://arxiv.org/abs/1608.07708`.

[3] Ekaterina Komendantskaya, John Power & Martin Schmidt (2016): *Coalgebraic logic programming: from Semantics to Implementation*. *J. Log. Comput.* 26(2), pp. 745 – 783.